Debugging is a skill that takes practice. There are a lot of different ways to debug a program, and I'll just talk through a few of them here, before giving you some buggy programs to practice fixing.

Syntax errors first Recognize if python is telling you that you have a syntax error, and look at Syntax errors if you need help. But be encouraged that syntax bugs are *always* shallow, in that it's just a matter of finding the typo.

Comment out code If you're not sure where an error is coming from (especially if it's a syntax error), it can help to comment out entire chunks of your code. In VS Code, you can do this by selecting code an hitting `ctrl-/`. This can rule out whole swathes of code that is not causing the error.

Run early and often This is less helpful for the activity I'm giving you today, but run your code as you write it, as soon as you have something that you don't expect to have a syntax error. This lets you catch bugs early, and gives you assurance that any new bug (esp. syntax errors) must be in the code you just wrote, rather than in code you wrote ten minutes ago.

Read your code Make sure to read your code carefully (esp. if you didn't write all of it yourself) to determine what you *think* it should do. If it doesn't do what you think it should do, then there is a bug that you haven't yet identified.

Add prints Adding calls to the `print` function allows you to test whether the code is doing what you think it is doing (see above). Finding the first deviation from what you expect gives you a strong hint as to where the bug might be.

Google for the error message Sometimes (but not always) searching for the error message will give you a good hint as to what went wrong.

Read the line numbers in a crash message These numbers are often unhelpful with syntax errors, but can be *extremely* helpful with runtime errors, as they tell you at which line of the code the problem arose.

Read the documentation If a function isn't having the effect you expect, try reading its documentation. Python documentation can be challenging to read, so keep in mind that this is just one approach to try.

Find example code If you're trying to do something with matplotlib or numpy, it can help to find a program that does something similar, and start with that program. This is often an effective fallback when reading the documentation fails.

Don't use a debugger It seems logical that a debugger would be useful for debugging. But they are not that useful. There may be times in your life when a debugger would help, but I wouldn't bet on it. (If you don't know what a debugger is, then you can ignore this.)

Read your code out loud to your partner Slowly reading the code out loud and having another person write down what it will do can catch persistent bugs. Our brains skip over things when we read quickly, and slowing down and forcing yourself to read each character can really help.

Run your code by hand Step through like we did on the board, and run your code on paper. You may find the bug, but you may also find values that you could print out, to test where things are going awry.

Explain the code to someone This is kind of like reading the code, but at a higher level. Looking through the code and explaining what it does can either reveal the bug to yourself, or it may be found by your listener. This is one of my favorite approaches: I'll get an undergraduate student or graduate student into my office and talk through my buggy code, and frequently it reveals the issue.

1 Your tasks

I have several buggy pieces of code. Some have a comment indicating what they are intended to do, and others should be clear from the code itself (e.g. a function named `factorial` should compute the factorial).

1. Debug code 1.
2. Debug code 2.
3. Debug code 3.
4. Debug code 4.
5. Debug code 5.
6. Debug code 6.
7. Debug code 7.